OPERATING SYSTEM-2 (GATE 2023) - REPORTS

OVERALL ANALYSIS | COMPARISON REPORT | SOLUTION REPORT

ALL(17) CORRECT(9) INCORRECT(7) SKIPPED(1)

Q. 1 Have any Doubt ?

Context switch times are dependent on

A. Number of registers to save and speed of data transfer for register to and from memory.

B. Memory management method used by operating system.

C. Speed of memory used in system.

D. All of the above — Your answer is **Correct**

Solution :
(d)
Registers and memory state need to save and restore on a context switch and they also affect the time of context switch.

QUESTION ANALYTICS +

Q. 2 Have any Doubt ?

Consider the following methods to pass parameters:
I. Pass the parameters in registers.
II. Store the parameters in a memory block/location and pass the address of this block/location in a register.
III. Push/place the parameters on the stack by sender and pop off the stack by receiver.
Which of the above methods can be used to pass parameters for system calls?

A. I

B. II, III

C. I, II

D. I, II, III — Your answer is **Correct**

Solution :
(d)
Which method and how many method used, is totally depend on the OS. And above given methods are generally used methods.

QUESTION ANALYTICS +

Q. 3 Have any Doubt ?

Consider the following information:
I. Page table base and limit/length register is paging used.
II. List of free-frames available in main memory if paging used.
III. I/O devices allocated to a process.
IV. Amount of CPU used by a process.
Which of the above given information is/are part of Process Control Block (PCB) of a process?

A. I and II

B. I, II and III

C. I, III and IV — Correct Option

Solution :
(c)
List of free-frames is maintained by OS and not stored in any PCB.

D. I, II, III and IV — Your answer is **IN-CORRECT**

QUESTION ANALYTICS +

Q. 4 Have any Doubt ?

Which of the following statement is incorrect?

A. In inter-process communication, message passing model is generally preferred over shared memory model for exchange of smaller amount of data.

B. Synchronous signals are delivered to the same process that performed the operation that caused the signal.

C. If the CPU scheduling is non-preemptive type then there can be the process state transition from running state to ready state. — Your answer is **Correct**

Solution :
(c)

In above transition the schedular preempt the running process and put that into ready state. Process itself cannot go to ready state. A process can go to terminate or block state by itself. So, it is preemption. But given CPU schedular is non-preemptive, a contradiction. So, option (c) is incorrect.

D. One thread immediately terminals the target thread in asynchronous cancellation method.

QUESTION ANALYTICS +

Q. 5 Have any Doubt ?

Which of the following is not an advantage of user-level thread over Kernel-level thread?

A. User-level thread context-switch is faster than Kernel-level threads switching.

B. User-level thread implementation can run on any operating system.

C. User-level thread scheduling can be application/process specific and independent of Kernel.

D. More than one user-level threads of a process can run concurrently. — Your answer is **Correct**

Solution :
(d)
User-level threads only managed by user space and Kernel is not aware about this. So, a single process can get only a single processor. And one processor can execute one thread only.

QUESTION ANALYTICS +

Q. 6 Have any Doubt ?

Consider the following two pseudo-code segments of two different processes:

```
Process-1                          Process-2
// some code                       // some code
int i = 0;                         int i = 0;
while (i < 10) {                   while (i < 10) {
        wait (S1);                         wait (S2);
    int x = m;                         int x = m;
    x = x + 1;                         x = x + 1;
    m = x;                             m = x;
    signal (S2);                       signal (S1);
    i = i + 1;                         i = i + 1;
}                                  }
// some code                       // some code
```

Where wait( ) and signal( ) are standard operations for binary semaphore. S_1, S_2 are binary semaphores. S_1 S_2 and m are global shared variable for Process-1 and Process-2. 'm' is a integer variable. Initial value of S_1, S_2 and m are 1, 0 and 0 respectively. Assume variables i, S_1, m, x and S_2 not used any where except given in codes. The value of m after finishing execution of above code segments is _____.

20 — Correct Option

Solution :
20
Initial values:

$$S_1 = 1, S_2 = 0 \text{ and } m = 0$$

So, P_1 will increment the m first, then it signal (S_2) for P_2, and wait for signal (S_1) in P_2. So, basically

they are incrementing m alternatively.
And both increment m 10 times each.
So, $m = 10 + 10 = 20$

QUESTION ANALYTICS +

Q. 7

Have any Doubt ?

Consider the following four processes with arrival time (in ms) and CPU burst time (in ms):

Process	Arrival time	Burst time
P_0	0	9
P_1	4	3
P_2	2	6
P_3	5	6

CPU uses Shortest Remaining Time First (SRTF) scheduling algorithm to schedule above process.
Average waiting time for execution of above process is _____ ms.

6

Your answer is Correct6

Solution :
6

Number of processes $(n) = 4$

Gantt chart for above process:

P_0	P_2	P_1	P_2	P_3	P_0

0 2 4 7 11 17 24

Waiting time = Finish T – Arrival T – Burst T

$WT(P_0) = 24 - 0 - 9 = 15$

$WT(P_1) = 7 - 4 - 3 = 0$

$WT(P_2) = 11 - 2 - 6 = 3$

$WT(P_3) = 17 - 5 - 6 = 6$

Average waiting time $= \frac{15+0+3+6}{4} = \frac{24}{4} = 6$

QUESTION ANALYTICS +

Q. 8

Have any Doubt ?

Consider a system using many-to-one pure user-level thread model in which Kernel is not aware of any user-level thread. Which of the following situations/states is/are possible for a process P with more than one user-level threads in above system?

A P is in running state and no thread of P is in running state. Your option is Correct

B P is in block state and a thread of P is in running state. Your option is Correct

C P is in running state and more than one threads of P are in running state. Your answer is IN-CORRECT

D P is in ready state and a thread of P is in running state. Your option is Correct

YOUR ANSWER - a,b,c,d | CORRECT ANSWER - a,b,d | STATUS - ✗

Solution :

(a, b, d)
In many-to-one model, Kernel is not aware of user-level-thread (ULT), so it consider any process as only a single process.
(c) Scheduling and context-switching of ULT processed by user libraries and process has only one processor, so one ULT possible in running state.
(a) According to Kernel if P is in running state, and when user do context switching in between ULT, then their is no thread is in running state.
(b), (d) if P and a thread in running state.
And some how P transferred to ready or block state then Kernel does not tell anything to user and ULT will remain in running state, but process state has changed. When P get again in running state then ULT will resume its execution.

QUESTION ANALYTICS +

Q. 9

Have any Doubt ?

When a process create a child, which of the following is/are correct possibility/situation?

A Parent process can execute parallely with children. Your option is Correct

B Parent process can wait until some or all of its children have terminated. Your option is Correct

B Parent process can wait until some or all of its children have terminated. Your option is Correct

C Child process can only be the duplicate of parent process. Your answer is **IN-CORRECT**

D Child process can only loaded with program other than parent program.

YOUR ANSWER - a,b,c | CORRECT ANSWER - a,b | STATUS - ✗

Solution :
(a, b)
Depend upon situation, parent process can execute parallely with children or after the children. Child process can be duplicate of parent process or loaded with other program.

QUESTION ANALYTICS +

Q. 10 Have any Doubt ?

Consider the following implementation/pseudo code of producer - consumer problem with buffer of size 5:

```
Process : Producer                  Process : Consumer
do {                                do {
    P(mutex);                           P(mutex);
    P(empty);                           P(full);
    /* access buffer to                 /* access buffer to
    add an item */                      remove one item */
    V(full);                            V(empty);
    V(mutex);                           V(mutex);
} while (true);                     } while (true);
```

Where mutex is binary semaphore, empty and full are counting semaphore. 'empty' represent number of unfilled places in buffer and 'full' represent number of filled places in buffer. Assume initially buffer has 3 times available.
Which of the following is incorrect?

A Deadlock is not possible in above implementation. Your answer is **Correct**

Solution :
(a)
Two possibilities of deadlock:
(i) If full = 0, mutex = 1, and consumer execute P(mutex) before producer, then producer will wait for V(mutex) in consumer and consumer wait for V(full) in producer.
(ii) If empty = 0, mutex = 1, and producer execute P(mutex) before consumer.
Access of buffer is mutual exclusive because of binary semaphore mutex.
Initially buffer has 3 items, so full = 3, empty = 5 – 3 = 2.

B Initial value of full is 3 and empty is 2.

C Both process producer and consumer cannot access buffer simultaneously.

D All of the above

QUESTION ANALYTICS +